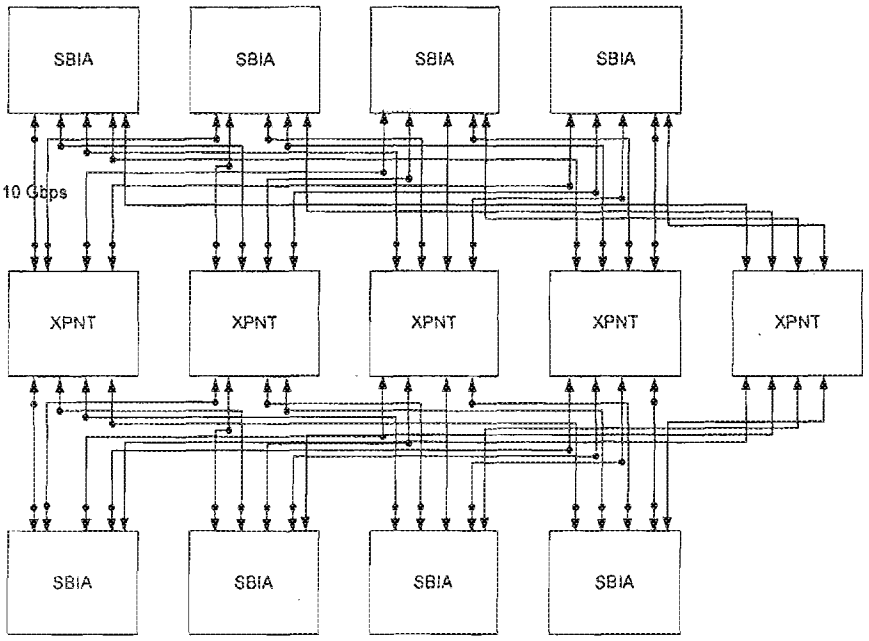
1 Introduction

This document describes a high level description of the Mucho Grande Architecture. It gives brief description of chips used as well as brief overview of functions within the chips. It also describes interfaces between major chips in the architecture. Further, it describes encoding schemes in the architecture as well as possible error condition in the arch. Lastly, it talks about flow control mechanism in the arch.

2 Overview

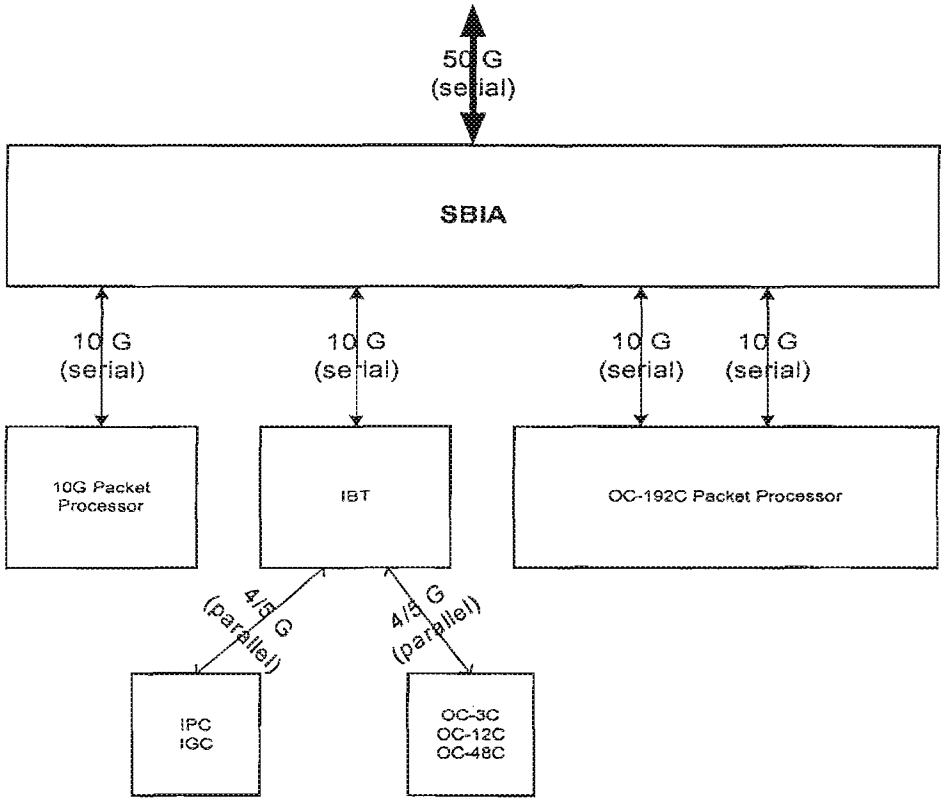
This architecture enables us to send 50 gigabit full-duplex traffic across the backplane. The backplane switches traffic from 8 different slots. In a slot, there is one SBIA chip, that sends and receives traffic to and from SXPNT chip. Each SXPNT chip is a 8-port crosspoint that receives and sends 10G of data from each slot. Data send and received from SBIA is striped across 5 SXPNT chips. The following diagram illustrates a 8-slot chassis where each SBIA represents 1 blade.



Serial link technology is used to send data across the backplane. Each serial link is capable of sending 2.5 Gbps of data who's baud rate is really 3.125 Gbps. Each serial link IP core offers a mechanism where by 4 serial links are group to create 1 10 Gbps pipe. The architecture manages to combine five 10G pipes into one 50G pipe. Thus the total switching capacity across the backplane is 50G x 8 x 2 =800 Gbps.

Having a 50G pipe to the backplane allows us to support 4 10G Ethernet traffic across all packet sizes at line rate or support 2 OC-192C at line rate or even one OC-768C.

As stated earilier, there is one SBIA per blade. This chip allows for combining 4 independent 10 G pipes into 1 50G pipe and for different traffic from the backplane to one of the four 10 G pipe. Further each SBIA is capable of supporting eight logical 4 Gbps pipes that is multiplexed across IBT chip. IBT acts as a bridge between exisitng BigIron ASIC (IPC/IGC) that make use of parallel I/O. It allows for converting parallel to serial and serial to parallel. The following diagram illustrates different blade configurations.



The above diagram tries to illustrate where different pieces would fit into the architecture using the basic Mucho chipset that consists of SBIA, SXPNT and IBT. In the case of OC-768C, there would be no need for SBIA. Although OC-192C shows taking up two 10G pipes, it could be made to use one. In this case, there would be performance impact to small packet sizes.

3 Super Backplane Interface Adapter (SBIA)

SBIA allows for a mechanism where by either four 10 pipes or eight 4/5 G pipes can be combined into 1 50 G pipe across the backplane. The ASIC can be configured to treat traffic from within a blade as either four 10 G pipes or as eight 5 gig pipes. The ASIC also performs local switching between the 4/8 local ports. On the traffic coming from the backplane, the ASIC synronizes data coming from 5 independent XPNT to create one 50 G line. It then goes through FID lookup to direct the packet to appropriate port. In the case of multi-cast traffic, the ASIC directs traffic to multiple ports based upon results obtained from FID lookup.

4 Super Crosspoint (SXPNT)

SXPNT is a 8-port crosspoint chip. It receives eight 10G streams of data with information to direct traffic to appropriate port within the data stream. The switch size is defined to be 1 cell. Each cell is defined to be either 8, 28, 48, 68, 88, 108, 128, or 148 bytes. Each SXPNT receives 1/5 of the data, in other words, 1/5 of the cell. Each cell is encoded in such a manner as to allow for all 5 SXPNT to operate indepentdly and for all SXPNT to receive destination slot number. Further details of backplane cell encoding will be described in later section. Each SXPNT is capable of 10 x 8 x 2 = 160 Gbps switching capacity.

5 IPC Bus Translator (IBT)

IBT ASIC acts as a bridge between IPC/IGC ASIC developed for BigIron Arch and Mucho Architecture. More generically, it allows for a mechanism to translate two 4/5 gig parallel stream into one 10G serial stream. The interface between SBIA and IBT will be described in later section. The parallel interface is the backplane interface of the IPC/IGC ASIC's. Additionally, IBT can be configured to work in the BigIron Architecture mode where a 10G serial design can be translated to BigIron 8G backplane interface and vise versa.

6 Transmission & Encoding Scheme for Mucho Grande Backplane

6.1 Introduction

This section describes a novel encoding scheme for the Mucho Grande backplane. The encoding requires use of 2 special K characters to communicate control information. The data sent from SBIA to sXPNT on

5 10G stripes, and from the sXPNT to SBIA on 5 10G trunks.

6.2 sBIA to sXPNT

The maximum size of a payload for transfer in the backplane is 160 bytes (148 bytes of data max, 10 bytes of "Start of Cell" (SOC) control information, and 2 bytes reserved. A complete 160 byte transfer, for the purpose of this document, is referred to as a "cell". A cycle is a single 3.2ns clock pulse (ie. 312.5 Mhz). A cell transfer is accomplished (as shown below) in 20 byte "blocks", in 8 consecutive cycles.

	Stripe 1				Stripe 2				Stripe 3				Stripe 4				Stripe 5			
cycle	L0	L1	L2	L3	L0	L1	L2	L3	L0	L1	L2	L3	L0	L1	L2	L3	L0	L1	L2	L3
1	K0	state	D0	D1	K0	state	D2	D3	K0	state	D4	D5	K0	state	D6	D7	K0	state	RES	RES
2	D8																			D27
3	D28																			D47
4	D48																			D67
5	D68																			D87
6	D88																			D107
7	D108																			D127
8	D128																			D147

6.2.1 Start of Cell

In the first block transmitted of a given cell, only 8 bytes are actually used for data. Of the remaining 12, 10 are used for control information, and 2 are reserved for debug purposes.

The K0 character is used by the sXPNT to recognize that a new cell has started, and that a state change may have occurred. The state byte is used by the sXPNT to determine what the destination slot for a given cell.

The "state" byte is assigned as follows:

Field	Name	Description
State[3:0]	SlotNumber	Destination slot number for sBIA to xXPNT and Source Slot Number for sXPNT to sBIA. sBIA will send IDLE packets to slot 7
State [5:4]	PayloadState	Encode payload state: 00 – RESERVED 01 - SOP 10 – DATA

		11 – ABORT
State [7]	Reserved	Reserved

In the case of transmission from sBIA to sXPNT, the value of "state" will be identical on all 5 stripes.

6.2.2 End of Packet

The scheme transfers 160 byte cells, whenever there are 148 bytes of data available to transfer. In the case where there are less than 148 bytes of data, then only as many 20 byte blocks as necessary are used. A special EOP K character is used to indicate an end of packet. There are 4 conditions that are of interest to us, and each is illustrated below:

1. EOP during cycle 1 (ie. during transmission of state information)

1	K0	state	D0	D1	K0	state	D2	D3	K0	state	K1	K1	K0	state	K1	K1	K0	state	RES	RES

Note that the K0, state, and Reserved bytes are all preserved, as in any other cycle 1 transmission. The K1 character is treated as data.

2. EOP during cycle n (n != 0)

1	K0	state	D0	D1	K0	state	D2	D3	K0	state	D4	D5	K0	state	D6	D7	K0	state	RES	RES
2	D8																			D27
3	D28				D32	D33	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1

3. EOP at block boundary during cycle n (n != 8)

1	K0	state	D0	D1	K0	state	D2	D3	K0	state	D4	D5	K0	state	D6	D7	K0	state	RES	RES
2	D8																			D27
3	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1	K1

Note that when n > 0, the block boundary for data is in lane 3 of stripe 5. However, for n = 0, the block boundary for data is in lane 3 of stripe 4.

4. EOP at cell boundary

6	D88																			D107
7	D108																			D127
8	D128																			D147

1	K0	state	K1	K1	K0	state	K1	K1	K0	state	K1	K1	K0	state	K1	K1	K0	state	RES	RES

Note that a cell need not be composed of 8 blocks, and can potentially be just one. The other exception to this rule is when an abort is send. ABORT is used during errornous condition in which case a cell could be less than 160 bytes and still not have an EOP. The very next cell for that destination will contain an ABORT cell.

6.2.3 Operation Concerns

Payload transmission size and method were both devised for very specific reasons.

6.2.3.1 Data Rate

In determining the ideal number of blocks in a cell, there are a number of constraints to consider. The sBIA design is optimized for transfers that are a multiple of 2 cycles long. Another constraint says that we would like to send as many pure "data" blocks per cell as possible, so as to mitigate the overhead lost to the cycle 1 block. Another constraint says that we would like to minimize the number of blocks per cell so as to more quickly arbitrate between the various devices connected to the sXPNT. Finally, the third, and most important, constraint says that we must run at line rate. A cell size of 6 blocks would only transmit 108 bytes of data, which causes the backplane data rate to drop below line rate for Ethernet packets of size 124 bytes. The next largest cell size available to us is 8 blocks. This leaves us with 7.5% of bandwidth lost to overhead, and allows for rapid arbitration between the various sBIA devices.

6.2.3.2 Reserved bytes

The cycle 1 block gives up half of its bandwidth to control information – a necessity for functionality. However, in stripe 5, we also "reserve" the 2 data bytes that were left, as well. This was done to simplify transfers from the existing packet processors. Both IGC and IPC are designed to transfer data in 4 byte increments. Therefore, to transmit and entire PPs payload, a multiple of 4 bytes must be available for data, or only a fraction of the payload will be sent. By reserving the 2 bytes of block 1, leave the number of available bytes as 8, and avoid any byte rotations. This is a non-issue when all 20 bytes are available for data.

6.3 sXPNT to sBIA

The constraints on the payload size for transfers from the xXPNT to the sBIA are identical to those for transfers in the opposing direction. However, there are major differences in the operation of the individual stripes.

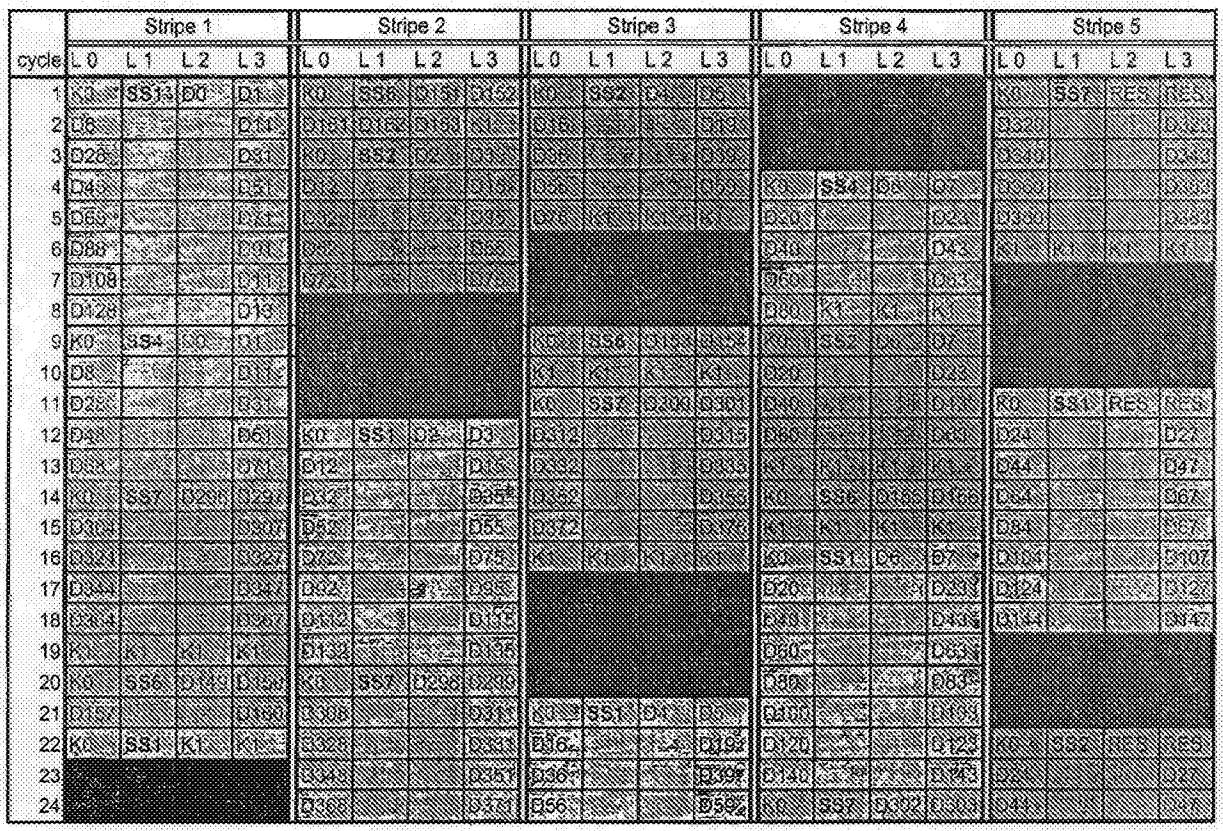
6.3.1 Cell Boundry Alignment

In the case of transfers from the sBIA to the sXPNT, all of the K0 characters were aligned within the same block. This was a simple task to accomplish as a single device was driving all five stripes. However, in this case, five devices are each driving a single stripe. While we are guaranteed to receive a K0 character at least every eight blocks for any given stripe, we have no guarantees regarding the alignment of the stripes them selves.

	Stripe 1				Stripe 2				Stripe 3				Stripe 4				Stripe 5			
cycle	L 0	L 1	L 2	L 3	L 0	L 1	L 2	L 3	L 0	L 1	L 2	L 3	L 0	L 1	L 2	L 3	L 0	L 1	L 2	L 3
1	K0	P1	D0	D1																
2	D8			D11																
3	D28			D31	K0	P1	D2	D3												
4	D48			D51	D12			D15					K0	P1	D6	D7				
5	D68			D71	D32			D35					D20			D23				
6	D88			D91	D52			D55	K0	P1	D4	D5	D40			D43				
7	D108			D111	D72			D75	D16			D19	D60			D63	K0	P1	RES	RES
8	D128			D131	D92			D95	D36			D39	D80			D83	D24			D27

6.3.2 Packet Alignment

In the example above, all five stripes are essentially transmitting the same packet, with a max skew of 6 cycles. In reality, since each sXPNT will be arbitrating among its sources independently of each other, not only can there be a skew in the cell boundary, but their can be as many as 28 cycles of skew between the transmission of a packet on one lane, versus its transmission on any other lane. This also means that other packets may be interlaced with our original packet, as well.



7 Mucho's Error Handling Capabilities

This section will describe possible error conditions that might occur with using serial link technology in the backplane and possible prevention care as well as recovery mechanism for it. It will also describe reset procedure to sync up different blades as well walk through events during hot swapping of blades.

7.1 Error Description

There are 3 types of errors that can occur in the backplane.

1. **Link Error.** This occurs as a result of a bit error or a byte alignment problem within a serdes. Since the clock is recovered from the data stream, there exists a possibility of a byte

alignment problem if there isn't enough data transition. Bit error can also occur as a result of external noise on to the line. The serdes also detected exception condition such as SOP character in lane 1 and marks them as link errors. The probability of either occurring is in the order of 1 failure per 10^^13 data bits transmitted.

2. **Lane Synchronization Error.** Lane is defined as 1 serial link among the 4 serial links that make up the 10 gig serdes. There exists a 4 deep fifo within the serdes cores to compensate for any possible transmission line skew and synchronize them as to present a unified 10 gig stream to the core logic. There are possible cases where the fifo's might overflow/underflow, which could result in lane synchronization error. There also exist cases when a lane synchronization sequence might determine a possible alignment problem. The probability of this occurring is unknown at this time. Broadcom is in the process of characterizing the serdes to determine the error rate.

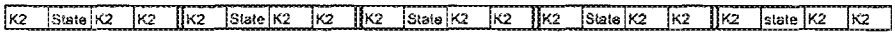
3. **Stripe Synchronization Error.** This is as a result of our architecture in which data is sent as 1 50 gig pipe to the backplane but is striped across 5 independent XPNT chips arbitrating indepently. The receiving BIA contains multiple 64 deep fifo's (156.25 Mhz) that is sorted according sending source and stripe. Once line fill across all stripes than a whole block "20 bytes" are read. There exist cases when 1 of the fifo's might overflow as a result of 1 of the above conditions or some other unknown case where the stripes might be completely out of sync. The probability of this is unknown at this time.

7.2 Detection and Preventive Care

1. **Link Error.** Once a link error is detected by the serdes, it passes an /E character through the parallel interface. This is detected by way of 8/10 decoder decoding an invalid character or a link being lost. After reset, the serdes go through a training sequence which allows the receiver to adjust it's sampling point to the middle of a bit time and through transition density offered through 8/10 encoding, the receiver in theory stays in sync. If however, the receiver's sampling point gets out of sync then at some point an invalid code is detected and passed through the parallel interface. The way to prevent the sampling point from going out of sync is to send random number /K and /R characters through the stream during idle operation. These K characters are special characters that allow the serdes to stay in sync by adjusting its sampling point. In the Mucho implementation, any time the link becomes idle than either a /K or /R will be transmitted. A pseudo random number generation will determine in selection of /K and /R characters.

2. **Lane Synchronization Error.** There are 2 ways that the serdes detect a possible lane alignment problem. This is detected when the fifo overflows/underflows. In this case, a signal is send from the serdes to the core logic when this occurs. The second failure is when serdes detect a failure when a lane synchronization sequence is sent across all lanes. In this case, the point at which data seen by the core going out of sequence is not known. The lane synchronization consists of send /A, /K, and /R characters send across all lanes simulatiounsly. Prior to lane synchronization, the serdes require finite number of random /K, /R characters to insure that the byte alignment exists. The serdes check to see if all these /A, /K, /R come across all lanes at once. Failure to detect all /A, /K, /R characters across all lanes not aligned is considered a lane synchronization error. The way to prevent the lanes from going out of sync is to periodically sent out /A, /K, /R sequence. In the Mucho implementation, there will be a programmable 32-bit counter that will run at 156.25 Mhz, that when it overflows it will send out the lane synchronization sequence. Further, there will be programmable padding register, 7-bit register, that will determine the number of random /K, /R characters to pad prior to and after /A, /K, /R sequence. We will work with Broadcom to determine what the production value of both of these registers values will be once the backplane is characterized.

3. **Stripe Synchronization.** There are couple of ways to detect possible stripe synchronization problem. First, if an invalid pattern is found across a block (block is defined a 5 stripes). An example of this would be detection of K0 pattern in lane 0 of 5 stripes and not across 1. This pattern would never be sent across the backplane and detection of such pattern is considered to be a stripe synchronization error. The other case is during stripe synchronization, if there are any entries left in the queue. Stripe synchronization will consist of sending a K2 character 64 times across all lanes and all stripes after which all stripes of the sync queues will write data to known location thus guaranteeing stripe synchronization. The number 64 is chosen since it will match the depth of the sync queues, thus, if there is any data left in the queue after the final K2 character is detected, this is considered a stripe synchronization error. In the mucho implementation, the BIA will send out the following pattern 64 times:



The state field is encoded with destination slot number as well as 1 bit used to tell whether you're in the middle of the stripe sequence or whether this is the last K2 transfer after which valid data follows. XPNT will pass this through it as if it were any data being passed.

7.3 Error Handling

The implementation will not differentiate between different types of errors described above and will treat all as violation error. This section will describe error process at each failure point and reasoning behind it. There are 4 places where an error is detected.

1. **XPNT -> BIA.** Error is detected on the receiving side of the BIA from one of the XPNT. When a link error or a lane error exists in one of the stripes, there exists a case when the determination of source sending the cell when an error occurred cannot be known since it could be between cell boundaries when the error is detected. Further, in the case of lane synchronization error, the determination of when a particular stripe loses synchronization cannot be known. For this reason, a general procedure is defined to handle link error and lane synchronization error. The procedure is as follows:
   a. The BIA will mark all packets from all slots that it has received a portion of the packet as being aborted. This will flush the packet down through the BIA onto the packet processor as AOP packet.
   b. It will reset its entire write pointer for sync queues and sync queue read pointers to a known value (0). This will insure stripe synchronization.
   c. Wait for the error condition to go away. No process will be defined to tell source slot from sending sync character.
   d. Wait for stripe alignment sequence across each slot. Once a sending source sends a stripe alignment character then the source slot is marked as in sync. This is needed since during an error the 5 stripes would have difficultly is knowing which SOP to sync to since packets of 40 bytes could exists thus a new packet every clock cycle. One could overcome this problem by having all packets tracking a sequence number but this requires at least 5-bit sequence number due to latency between stripes within the XPNT. The 5 bits don't exist. This is a much more robust and a much simpler solution.
   e. Wait for SOP following the slot being marked as striped aligned.

   In the case of a stripe alignment error, there is no need to flush all the slots. Thus only the slot with a stripe alignment error is put through the above process.

2. **BIA->XPNT.** Error is detected on the receiving side of the XPNT. There are 2 types of problem that can exists, link error and lane synchronization error. At first glance, it seems like a simple problem to solve, send an AOP and you're done. However, this poses an

interesting problem on the receiving BIA in that the 5 independent XPNT's now have different number of input to arbitrate from. An example of this is a scenario is when slots 0, 1, and 2, are transmitting to slot 7 and slot 0's 5th stripe link goes down. Now in the scenario, stripe 0 to 4 will arbritate between slots 0, 1, and 2 while the 5th stripe will arbitrate between only slots 1, and 2 for destination slot 7. This results in possible temporary overflow of sync queue for slot0 for stripes 0 to 4 on destination slot 7 and overflow of slot1 and slot2 for stripe 5 for destination slot 7. This problem manifests to all slots if there is enough traffic onto the backplane and link is down for long time. The receiving XPNT that detects the error cannot know destination slot of the data when an error occurred similar to receiving BIA getting a link error or lane error. For this reason, it must flush all it destination queues from the bad source slot. The following procedure will be followed by the XPNT in case of an error:

a. Send an AOP to all slots.
b. Wait for error to go away.
c. Sync to K0 token after error goes away to begin accepting data.

On the receiving BIA side, when an AOP is detected from a particular stripe, then the process similar to one of the stripe being out of sync is followed. This could also result in stripe synchronization error on different slots as described with example above in which case stripe synchronization procedure for BIA is followed.

Since the probability of such failures is unknown at this time, the following OPTIONAL signals will also exists that need not be hooked up on the board. There will be a set of error ready's from each XPNT to BIA that will be wired or on the board and hooked to BIA. The motivation for this is when an error is detected by on of the XPNT, the XPNT send stop the BIA from sending any more data. This prevents huge amounts of data from being dropped when an error exists. It will also prevent sync queues from overflowing on the receive side of the BIA. The sending BIA will continuously repeat lane synchronization sequence until an error signal is deasserted. After deassertion, it will send out stripe synchronization sequence to all slots. The other OPTIONAL set of signals that were considered but determined to be fatal were flow control from BIA to XPNT based on the sync queues reaching a certain threshold. The motivation behind this was to prevent loss of data due to momentary overflowing of queues when BIA->XPNT failure occurs. This was considered fatal in that there could exists cases where one of the stripes would overflow while one of them is empty. This could result in a hang condition. To prevent this from occurring the flow control mechanism must consists of flow control based on source slot as well as individual stripes. Due to huge amount of pin requirement of XPNT (56), as well as huge pin requirements on BIA (35), if this option is to be considered it must be some sort of a serial backpressure.

3. **IBT->BIA.** Error detected on the receiving side of the BIA is treated identical to error receiving on the XPNT from the BIA. Since as with other errors described above, since the destination slot cannot be know under certain conditions by the BIA, the following process is followed:
   a. Send an AOP to all slots.
   b. Wait for error to go away.
   c. Sync to K0 token after error goes away to begin accepting data.

4. **BIA->IBT.** Error detected on the receiving side is the IBT is treated identical to error seen by the BIA from IBT. The following process will be used.
   a. Send an AOP to all slots of down stream IPC/IGC to terminate any packet in progress.
   b. Wait for error to go away.
   c. Sync to K0 token after error goes away to begin accepting data.

7.4 Reset Procedure

The following reset procedure will be followed to get the serdes in sync. An external reset will be asserted to the serdes core when a reset is applied to the core. The duration of the reset pulse for the serdes need not be longer than 10 cycles. After reset pulse, the transmitter and the receiver of the serdes will sync up to each other through defined procedure. It is assumed that the serdes will be in sync once the core comes out of reset. For this reason, the reset pulse for the core must be considerably greater than the reset pulse for the serdes core.

The core will rely on software interaction to get the core in sync. Once the BIA, IBT, and XPNT come out of reset, they will continuously send lane synchronization sequence. The receiver will set a software visible bit stating that it's lane is in sync. Once software determines that the lanes are in sync, it will try to get the stripes in sync. This is done through software which will enable continuously sending of stripe synchronization sequence. Once again, the receiving side of the BIA will set a bit stating that it's in sync with a particular source slot. Once software determines this, it will enable transmit for the BIA, XPNT and IBT.

7.5 Hot Swapping

Hot Swapping will result in a link error on the XPNT. This will be equivalent to XPNT->BIA link going down. The same process will be followed. When a new blade is plugged in, it will require a software walk through as described in the reset procedure to bring the new blade to life.

8 IBT to SBIA Encoding Scheme

IBT transmits and receives packets to and from SBIA through XAUI interface. Packets are segmented into cells which consist of a four byte header followed by 32 bytes of data. End of packet is signaled by K1 special character on any invalid data bytes within four byte of transfer or four K1 on all XAUI lanes. Each byte is serialized onto one XAUI lane.

Lane3	Lane2	Lane1	Lane0
Reserved	Reserved	State	K0
D3	D2	D1	D0
D7	D6	D5	D4
D11	D10	D9	D8
D15	D14	D13	D12

Data packets are formatted into data cells which consists of a header plus a data payload. 32 bit of header takes one cycle on four XAUI lanes. It has K0 special character on Lane0 to indicate that current transfer is a header. The state information will go on Lane1 of a header.

Field	Name	Description
State[3:0]	SlotNumber	Destination slot number from IBT to SBIA. IPC can address 10 slots(7 remote, 3 local) and IGC can address 14 slots (7 remove and 7 local)
State [5:4]	PayloadState	Encode payload state: 00 – RESERVED 01 - SOP 10 – DATA 11 – ABORT
State[6]	Source/Destination IPC	Encode source/destination IPC id number 0 – to/from IPC0 1 – to/from IPC1

State [7]	Reserved	Reserved

9 Flow Control

IBT to SBIA flow control:

IBT flow controls SBIA on a per IPC basis through two dedicated pins, sds_rxready[1:0].
IBT de-asserts sds_rxready[1:0] under two conditions:

1) IBT internal FIFO for IPC0/1 is filled above it's high water mark.
2) IPC0/1 de-asserts its bp_rxready to IBT

SBIA to IBT flow control:

SBIA flow controls IBT on a per IPC/IGC per destination slot basis. It serializes 14 ready status to IBT through a periodical sync and two txready signals for each IPC. IBT decodes it and passes on to IPC0/1 in 6 cycles. IPC and IGC system will have the same clock cycle latencies.

sync	Tx_ready[3:2]	Tx_ready[1:0]	Descriptions
1	IPC1 Slot[1:0]	IPC0 Slot[1:0]	Slot[1:0] ready to receive from IPC/IGC/0/1
0	IPC1 Slot[3:2]	IPC0 Slot[3:2]	Slot[3:2] ready to receive from IPC/IGC/0/1
0	IPC1 Slot[5:4]	IPC0 Slot[5:4]	Slot[5:4] ready to receive from IPC/IGC/0/1
0	IPC1 Slot[7:6]	IPC0 Slot[7:6]	Slot[7:6] ready to receive from IPC/IGC/0/1
0	IPC1 Slot[9:8]	IPC0 Slot[9:8]	Slot[9:8] ready to receive from IPC/IGC/0/1
0	IPC1 Slot[11:10]	IPC0 Slot[11:10]	Slot[11:10] ready to receive from IPC/IGC/0/1
0	IPC1 Slot[13:12]	IPC0 Slot[13:12]	Slot[13:12] ready to receive from IPC/IGC/0/1